**Homework Assignment #2 (100 Points)**

**CoLab, GitHub, Python Libraries, DFS, and BFS**

This homework will allow you to learn how to use Google Drive and GitHub in conjunction with Google Colab and how to use different Python libraries for image processing. Some of the libraries are described in Homework #1. For example, there are useful Python libraries for image processing. Please get yourself familiar with those essential Python libraries.

1. Create your own Google account, Google Drive, and GitHub account. Your Google Drive and GitHub may be used for other homework and your course project. Show that you already have your Google Drive and GitHub. (10 Points)
2. Demonstrate how to link Google CoLab with your Google Drive and GitHub. Show how to access the files/folders in your Google Drive from Google CoLab. Show how to access the files/folders in your GitHub from Google CoLab. (15 Points)
3. Write some Python codes to demonstrate that you are able to read some image files from your Google Drive and then process those images. For example, demonstrate how to apply a low-pass filter to a noisy image, how to apply a median filter to an image, or how to perform histogram equalization processing on an image. Refer to Homework 1 – Section 6.2 for the list of image processing functions. In your homework report, please show your codes with explanations, the original images, the processed images, and the descriptions/explanations of the results. (25 Points)
4. Implement Depth First Search (DFS) algorithm in Python. Refer to the DFS content that is described in the lecture, create a new graph that is different from the graph in the lecture. The graph must have four layers and at least 20 nodes. Implement the DFS algorithm in Python. See some sample codes in Appendix 1 below. Apply the algorithm to the graph that you newly created. In your report, please include the graph that you created, the codes with explanations, the explanations of the output result from the Python algorithm. Please manually do the depth first search and verify whether the output result of the Python algorithm is correct. (25 Points)
5. Implement the Breadth First Search (BFS) algorithm in Python. Refer to the BFS content that is described in the lecture, create a new graph that is different from the graph in the lecture. The graph must have four layers and at least 20 nodes. Implement the BFS algorithm in Python. See some sample codes in Appendix 2 below. Apply the BFS algorithm to the graph that you newly created. In your report, please include the graph that you created, the codes with explanations, the explanations of the output result from the Python algorithm. Please manually do the breadth first search and verify whether the output result of the Python algorithm is correct. (25 Points)

Please submit the homework through eLearning -> Homework -> Homework 2. Your homework should include your name, homework # (Homework 2), codes with explanations, results, and descriptions of the results. The due date of the homework report is **Tuesday, September 3, 2024, at 11:59 PM**.

Refer to the appendix below for the sample codes.

**Appendix 1: Sample Python codes for the depth first search algorithm.**

(Note: Different Phyton versions may be different. The codes are only used a reference)

# Using a Python dictionary to act as an adjacency list

graph = {

'A' : ['B','C'],

'B' : ['D', 'E'],

'C' : ['F'],

'D' : [],

'E' : ['F'],

'F' : []

}

visited = set() # Set to keep track of visited nodes.

def dfs(visited, graph, node):

if node not in visited:

print (node)

visited.add(node)

for neighbour in graph[node]:

dfs(visited, graph, neighbour)

# Driver Code

dfs(visited, graph, 'A')

**Appendix 2: Sample Python codes for the breadth first search algorithm.**

(Note: Different Phyton versions may be different. The codes are only used a reference)

# Using a Python dictionary to act as an adjacency list

graph = {

'A' : ['B','C'],

'B' : ['D', 'E'],

'C' : ['F'],

'D' : [],

'E' : ['F'],

'F' : []

}

visited = [] # List to keep track of visited nodes.

queue = [] #Initialize a queue

def bfs(visited, graph, node):

visited.append(node)

queue.append(node)

while queue:

s = queue.pop(0)

print (s, end = " ")

for neighbour in graph[s]:

if neighbour not in visited:

visited.append(neighbour)

queue.append(neighbour)

# Driver Code

bfs(visited, graph, 'A')